Targeting FPGA DSP Slices for a Large Integer Multiplier for Integer Based FHE

Targeting FPGA DSP Slices for a Large Integer Multiplier for Integer Based FHE

Targeting FPGA DSP Slices for a Large Integer Multiplier for Integer Based FHE

Ciara Moore, Neil Hanley, John McAllister, Máire O'Neill, Elizabeth O'Sullivan and Xiaolin Cao

Centre for Secure Information Technologies (CSIT),
Queen's University Belfast, Northern Ireland
cmoore50@qub.ac.uk, n.hanley@qub.ac.uk,
j.Mcallister@ecit.qub.ac.uk, m.oneill@ecit.qub.ac.uk,
e.osullivan@qub.ac.uk,xcao03@qub.ac.uk

Abstract. Homomorphic encryption offers potential for secure cloud computing. However due to the complexity of homomorphic encryption schemes, performance of implemented schemes to date have been unpractical. This work investigates the use of hardware, specifically Field Programmable Gate Array (FPGA) technology, for implementing the building blocks involved in somewhat and fully homomorphic encryption schemes in order to assess the practicality of such schemes. We concentrate on the selection of a suitable multiplication algorithm and hardware architecture for large integer multiplication, one of the main bottlenecks in many homomorphic encryption schemes. We focus on the encryption step of an integer-based fully homomorphic encryption (FHE) scheme. We target the DSP48E1 slices available on Xilinx Virtex 7 FPGAs to ascertain whether the large integer multiplier within the encryption step of a FHE scheme could fit on a single FPGA device. We find that, for toy size parameters for the FHE encryption step, the large integer multiplier fits comfortably within the DSP48E1 slices, greatly improving the practicality of the encryption step compared to a software implementation. As multiplication is an important operation in other FHE schemes, a hardware implementation using this multiplier could also be used to improve performance of these schemes.

1 Introduction

Cloud computing offers numerous advantages to users, such as computing as a service, storage and management of large amounts of data. Yet this requires the trust of the public cloud service provider to maintain an adequate level of security and prevent leakage of private data. Data security has been shown to be the greatest concern of clients who use the cloud [1]. If users could encrypt their data before storing it in an (untrusted) cloud server and still be able to compute on these ciphertexts, they could take advantage of the benefits of cloud computation without the risk of leaking their private data.

Secure cloud computing could be achieved by the use of an efficient fully homomorphic encryption scheme. Homomorphic encryption is a method of encryption featuring four steps: {*key-gen, encrypt, evaluate, decrypt*}, where the step *evaluate* enables the correct computation, such as addition and multiplication, on ciphertexts without the use of decryption. Traditionally, homomorphic encryption schemes were either additively or multiplicatively homomorphic; such schemes are also known as partially homomorphic encryption schemes. Examples include the multiplicatively homomorphic ElGamal [2] and the additively homomorphic Paillier [3] cryptosystems. In 2005 Boneh-Goh-Nissam introduced a scheme which allowed a combination of additions and one multiplication on encrypted data [4].

The area of homomorphic encryption leapt forward in 2009 however, with Gentry's ground-breaking work on a *fully* homomorphic encryption (FHE) scheme based on ideal lattices, which introduced the first technique to allow an arbitrary number of operations (both additions and multiplications) to be employed on ciphertexts [5]. A FHE scheme is created by extending a *somewhat* homomorphic encryption (SHE) scheme, which allows a limited number of multiplications and additions. In the last few years there has been much research to improve the efficiency of homomorphic encryption schemes [6], [7], [8]. The theory behind homomorphic encryption is developing at a quick pace; however there are few published results of timings from implementations of these schemes. Moreover, from the results that have been published, it is clear that improvements in the efficiency of these schemes are still needed. For example, in the SHE implementation of the largest lattice-based scheme in [9], bitwise encryption is reported to take 3.2 minutes. In addition, the FHE implementation of the integer-based scheme for the large implementation in [10], bitwise encryption takes 7 minutes 15 seconds. The recent FHE implementation of AES [8] requires approximately 36 hours and 256 GB RAM to evaluate AES; this shows there is still much to be done before such schemes are practical and comparable to existing cryptographic encryption schemes. It also highlights the complexity of homomorphic encryption and underlines the demand for more efficient implementations. In this paper we investigate implementing a hardware building block, which in some form features in all of the SHE and FHE schemes, in order to improve their performance and hence their practicality.

Three main structures have been proposed for FHE/SHE schemes: lattice-based, integer-based and schemes based on learning with errors (LWE) or ring learning with errors (RLWE). The current focus of the research community is on RLWE schemes, as these promise greater efficiency due to recent optimisations to support batching, for example in [7]. However the integer-based schemes, introduced by van Dijk, Gentry, Halevi and Vaikuntanathan (DGHV) in [11], have a relatively simple structure in comparison to the RLWE schemes and lattice-based methods introduced by Gentry. The efficiency of the latest integer-based schemes [10], [12] is comparable to the lattice-based schemes.

As a first step in our investigation into a hardware implementation of SHE or FHE schemes, we consider the proposed parameter sizes and the main underlying computations involved in the encryption step of the integer-based FHE scheme proposed by Coron et al [10], a scheme similar to the original DGHV integer-based FHE scheme

[11]. The main computations are modular reduction and large integer multiplication, and are used in all of the FHE schemes. Therefore an efficient hardware implementation of these crypto-primitives can be used in future real time hardware implementations of any FHE scheme to improve performance. We focus on considering a hardware implementation of large integer multiplication and highlight some of the major issues involved. We begin to address these implementation issues by selecting a suitable large integer multiplication algorithm for hardware implementation. Due to the computational complexity of large integer multiplication, it is likely that a custom circuit architecture exploiting an Application Specific Integrated Circuit (ASIC) or a high-end FPGA technology in the form of a Xilinx Virtex 7 device will be required to enable real-time implementation. Considering the reconfigurable nature and quick development time of FPGAs we base our implementations on these. These devices also have exceptional levels of on-chip multiplication capability in the form of DSP48E1 slices.

To our knowledge, there are no current hardware implementations of complete FHE schemes; however there has been work on FPGA implementation of primitives for a SHE scheme using Mathworks® Simulink [13]. There has also been research in similar areas, for example [14] discusses the practicality of existing applications of homomorphic encryption by an empirical evaluation based on the lattice-based scheme by Smart and Vercauteren [15], and highlights implementation issues such as memory access. Another related publication [16] considers the hardware building blocks for the LWE cryptosystem and uses Fast Fourier Transform (FFT) multiplication in polynomial rings. Although it is stated that there may be more suitable multiplication algorithms for this purpose, it is shown that this hardware implementation of LWE still outperforms the software implementation. The Comba multiplication algorithm, introduced in 1962 [17], has been implemented in an FPGA using DSP slices to carry out multiplications required in the area of elliptic curve cryptography [18]. We look at using this multiplication method for large integer multiplication required in FHE schemes, as this type of multiplication has been shown to be very suitable for use on DSP slices. We estimate the performance of using Comba multiplication in DSP slices for the parameter sizes in the integer-based scheme by Coron et al [10] in order to establish the feasibility of a FPGA implementation of FHE schemes, and whether a hardware implementation of a multiplier would enable practical performance of the encryption step in [10], therefore offering a significant improvement to the existing implementations of large integer multiplication in FHE schemes.

We find in this initial evaluation for the toy-sized version of the encryption step of the FHE scheme in [10], the large integer multiplier fits comfortably within the DSP48E1 slices in a FPGA and would improve the practicality of the encryption step in [10], compared to a software implementation. Moreover, the large integer multiplier for the specified small, medium and large versions of the encryption step also fits comfortably within the DSP48E1 slice, though in these versions off-chip memory must be used to cope with the large parameter sizes. Indeed, as multiplication is an important operation in this type of encryption scheme, a hardware implementation using this multiplier, could be used to improve the performance of all FHE schemes.

To our knowledge, there has been little previous analysis into the practicality of an FPGA based implementation of crypto primitives for FHE schemes.

In Section 2 of this paper, the selected integer-based scheme is introduced and we justify our approach. Section 3 presents a very brief survey of some multiplication methods and introduces the Comba multiplication method. A suitable hardware architecture and rough estimates for timings and resource requirements is given in Section 4. Some of the major implementation issues are also highlighted in this section.

2 Overview of Integer-Based FHE Scheme by Coron et al.

We focus on the proposed FHE scheme by Coron et al [10], based on the original integer-based FHE scheme [11], for its simple approach, detailed parameter sizes and reasonable performance in comparison to other implemented schemes, such as [9], [15]. We focus in particular on the encryption step, as this is one of the key steps in a FHE scheme which may need to be performed multiple times, unlike key generation which is only required initially. Moreover the encryption step in [10] involves two important cryptographic building blocks: multiplication of large integers and modular reduction, which are also used in all other FHE schemes. We explain the encryption step in the integer based FHE scheme in detail because of its relevance to this work. However, we refer the reader to [10] for details of the other steps in the scheme.

The encryption step for a given message $m \in \{0,1\}$ is given as:

$$c \leftarrow m + 2 \cdot r + 2 \cdot \sum_{i=1}^{\tau} b_i \cdot x_i \; mod \; x_0 \tag{1}$$

where r is an integer from a specified range $(-2^{\rho'}, 2^{\rho'})$ and is used as random noise; $x_0 = q_o \cdot p$, where q_0 is a random odd integer in the range $[0,2^{\gamma}/p)$ and p is a random prime integer of η bits; x_i for $1 \leq i \leq \tau$ is an array of large random integers; and b_i, $\forall 1 \leq i \leq \tau$, is an array of random integers selected from a smaller range $[0,2^{\alpha})$. The parameters γ, ρ', η and α in Equation (1) vary according to the size of scheme implemented. Hence we refer the reader to [10] for full details on these parameters and further information on the generation of x_i.

We target in particular the toy-sized FHE scheme; the parameter sizes for the four versions of the FHE scheme are listed in Table 1. In the toy-sized scheme 158 multiplications of $b_i \cdot x_i$ are required where the bit sizes for b_i and x_i are 936 bits and 150,000 bits respectively. In this paper we focus on the multiplier and establish a suitable approach to deal with these large parameter sizes. As can be seen in Table 1, the parameter sizes are very large, which is common in FHE schemes. For a discussion of security of this scheme, we again refer the reader to [10].

Table 1. Parameter Sizes (bits) for Encryption step in FHE Scheme in [10]

Parameter	Toy	Small	Medium	Large
b_i	936	1,476	2,016	2,556
x_i	150,000	830,000	4,200,000	19,350,000
x_0	150,000	830,000	4,200,000	19,350,000
τ	158	572	2110	7659

The two main bottlenecks in the selected scheme are large integer multiplication and modular reduction. These operations are also required in many other FHE schemes, such as the lattice based schemes [9], [15]. We have chosen to focus initially on multiplication as most efficient hardware implementations of modular reduction also require the use of a multiplier, for example Barrett reduction and Montgomery reduction both require multiplications [19]. Moreover, one of the main motivations for FHE and SHE schemes is to compute, using additions and multiplications, on encrypted data. Therefore an efficient multiplier for large parameter sizes is essential for such schemes.

Multiplication is only one of the issues to be addressed to implement this type of encryption scheme in hardware. Other major issues in the hardware implementation of homomorphic encryption schemes exist, such as the transfer of large blocks of data to and from the board, memory access and efficient scheduling of operations. In this initial study, we focus our attention on the multiplication bottleneck to establish the viability of an FPGA implementation of a FHE scheme and thus to justify continuing research to address the other important issues for a hardware implementation.

3 Overview of the Comba Multiplication Algorithm

Many multiplications with large multiplicands are required for implementation of the selected encryption scheme. There are various different algorithms available to deal with larger multiplicands and multipliers. Karatsuba multipliers [20] can be used to reduce the number and size of multiplications for large numbers by representing the large numbers, X and Y, as additions of two smaller numbers, for example $X = X_1 2^k + X_0$, $Y = Y_1 2^k + Y_0$ where X and Y are numbers of bit length $2k$. Then the multiplications are reduced from 4 multiplications (and 3 additions) to 3 multiplications (and 1 addition and 3 subtractions) as shown in Equation (2):

$$XY = (X_1 2^k + X_0) \cdot (Y_1 2^k + Y_0)$$
$$= 2(X_0 \cdot Y_0 + X_1 \cdot Y_1 2^{2k}) - (X_1 2^k - X_0) \cdot (Y_1 2^k - Y_0) \quad (2)$$

However, Karatsuba requires intermediate storage of multiplication and subtraction results and is therefore not ideal for mapping to DSP slices, especially when considering such large parameter sizes. Fast Fourier transforms (FFTs) can also be used for multiplications, particularly when many multiplications are required. The use of FFTs has also been suggested in previous homomorphic encryption implementations [13]. Another alternative is Montgomery multiplication, commonly used in asymmetric cryptosystems. However, this technique requires multiplications for both post- and pre-computation. This method is more suitable when repeating multiplications such as in exponentiation algorithms, for example in RSA [21]. As we propose to target the DSP slices on a FPGA for large integer multiplication, we select a multiplication algorithm particularly suitable for the underlying FPGA platform for our initial investigation. The Comba multiplication method introduced in [17] is used for hardware-based large integer multiplication in [18] and it is very suitable for use on DSP slices as it can be easily broken down into partial products, therefore making efficient use of

resources. Moreover, when these partial products are accumulated, they are retained within the DSP block. This method of multiplication involves a reversal of the order of words in the multiplicand, several shifts and multiplications with each shift. For example, to multiply two 3-word numbers, $A \times B$ for $A = A_2A_1A_0$ and $B = B_2B_1B_0$, reverse B => $B' = B_0B_1B_2$ and calculate the partial products PP_i by multiplying and adding:

$$\begin{aligned} PP_0 &<= A_0 \times B_0 \\ PP_1 &<= A_1 \times B_0 + A_0 \times B_1 \\ PP_2 &<= A_2 \times B_0 + A_1 \times B_1 + A_0 \times B_2 \\ PP_3 &<= A_2 \times B_1 + A_1 \times B_2 \\ PP_4 &<= A_2 \times B_2 \end{aligned}$$

Each of the partial products PP_i are shifted left by i words ($\ll i$) and summed together to give the final product, giving:

$$A \times B = (PP_4 \ll 4) + (PP_3 \ll 3) + (PP_2 \ll 2) + (PP_1 \ll 1) + PP_0.$$

For a generalised multiplication of $A \times B$, let the word-length of A equal m and the word-length of B equal n and without loss of generality let $m \geq n$. There will be $m + n - 1$ required partial products in the Comba multiplication. When $i < m$, the i^{th} partial product PP_i requires $i + 1$ multiplications. The partial products can therefore have a maximum of m multiplications. When $i \geq m$, the i^{th} partial product requires $m + n - 1 - i$ multiplications. As suggested in [18] we can combine the partial products into m steps which have m multiplications in each step. Continuing the above example, we then have three steps which combine all of the partial products:

$$\begin{aligned} PP_0' &<= A_0 \times B_0 \quad \boldsymbol{A_2 \times B_1} + \boldsymbol{A_1 \times B_2} \\ PP_1' &<= A_1 \times B_0 + A_0 \times B_1 \quad \boldsymbol{A_2 \times B_2} \\ PP_2' &<= A_2 \times B_0 + A_1 \times B_1 + A_0 \times B_2 \end{aligned}$$

We refer the reader to [18] for further details on this optimisation and their hardware implementation.

The choice of multiplier greatly depends on the size of the multiplication. In the particular case of the implementation of the toy scheme mentioned previously, we have a multiplier of 936 bits and a multiplicand of 150000 bits. We therefore propose to use a 936 bit multiplier and this can then be used several times and the partial products can be added to achieve the overall large multiplier. When we consider an FPGA implementation of Comba multiplication, we can run each of the steps in a separate parallel DSP slice, and then the number of clock cycles required per multiplication is m, the number of words in the largest multiplicand, and a few extra clocks for the summation of the partial products. The number of DSP slices required for the multiplication is equal to the number of steps after combining the partial products which is also m.

4 DSP Slice Usage and Estimated Timings for Large Integer Multiplier

FPGAs are a suitable target technology for hardware for implementations of SHE and FHE. They are cheaper and offer greater flexibility than ASIC devices. This makes them suitable for cryptographic purposes, as they can be re-programmed in-situ when protocols are changed and updated. The latest FPGA devices offer a large amount of embedded hardware blocks, which can be used to carry out optimised operations, such as addition and multiply-accumulate steps. The inclusion of dedicated DSP slices on an FPGA allows for very efficient multiplication and multiply-accumulate (MAC) operations. For example, on current Xilinx Virtex 7 FPGAs there are up to 3600 DSP48E1 slices, each with the capacity of a 18×25 bit signed multiplication and 48 bit accumulation; see Table 2 for further examples. Furthermore, the 18×25 bit signed multiplier and the 48bit accumulator are capable to run at frequency of up to 741 MHz [22].

Table 2. Examples of Available DSP Slices in Selected Xilinx Virtex 7 FPGA Devices

Xilinx Virtex 7 FPGA	No. of DSP Slices	No. Columns
XC7VX415T	2,160	18
XC7VX485T	2,800	20
XC7VX980T	3,600	20

The compact size imposes significant constraints on input word sizes and storage, which is problematic for FHE and SHE implementations with large key and cipher-text sizes. To circumvent this in so far as possible, we target one of the largest FPGAs, the Xilinx Virtex 7XC7VX980T.

Our goal is to implement a 936 bit multiplier. An un-optimised initial inference of a multiplier using ISE Design Suite reveals that a 936 bit multiplier requires 76% of the targeted device's DSP slices. Therefore this highly un-optimised large multiplier fits on to the FPGA device. However, inferring and cascading multipliers in ISE Design Suite requires exponentially many DSP slices for increasingly large multiplications. If an un-optimised implementation of a 936 bit multiplier is designed using partial products and shifts for example, this will most likely occupy over 2700 DSP slices just to implement the toy-sized multiplier. The Xilinx Core Generator can only generate multipliers for up to 64 bits long, which is much smaller than our required multiplier. Therefore an indirect approach must be taken.

In [18] the dedicated hard core functions on FPGAs are targeted to produce efficient implementations of both AES and ECC. An efficient multiplier is presented, which firstly calculates the partial products using the Comba method and then these are added to generate the final result. This technique is very suitable for large integers, as the DSP slices can be used in parallel, which allows for less device usage for the same multiplier size. Although our target multiplier is 936 bits, we use a 16 bit unsigned multiplier, as the DSP slice has an 18-bit signed multiplier and thus a 16 bit multiplier is the most suitable size to work with that fits within a DSP slice. Using

this approach, a 944 bit multiplier can be designed using 59 DSP slices, where a few of the multiplications are redundant. Each of these 59 DSP slices will calculate 16×16 multiplications up to 59 times and thus a full 944 bit multiplication can be calculated in around 60 clock cycles. This multiplier can then be run multiple times to reach the appropriate multiplication size. Therefore the 150000 bit x_i is represented in 9575 16 bit words and the 944 bit multiplier is used approximately 159 times. After each multiplication in the DSP slice the x_i are shifted right, and further multiplications with the shifted x_i are carried out and accumulated in the DSP slices. The partial product adder combines these partial products. The least significant word of each of the partial products accumulated in the DSP slices is saved in a register and the remainder is added to the next partial product. This process is continued with all of the partial products consecutively to give the final output. Additionally, several multipliers of this size can be implemented in parallel to increase the performance of the multiplier in the encryption step.

Figure 1 shows a basic hardware architecture design of the Comba multiplier as proposed in [18]. The chosen 944 bit multiplicand and multiplier are both represented by 59 16 bit words, as shown by registers A and B in Figure 1; A and B represent the x_i and b_i from the selected FHE scheme. The value s is equal to the number of words, in this case 59, and w is equal to the word size, 16. This can be extended to larger sizes: $s = Multiplier\ Size\ /\ Word\ Size$. Each of these 59 words from both A and B is input into a separate DSP slice, again as shown in Figure 1. The product of these two terms is accumulated within the DSP slices, using the internal 48 bit accumulator logic. The accumulation output is a maximum of $2w + \log_2(s)$ bits. After each multiplication, the x_i in Figure 1 are shifted left by 16 bits and a new word is input to each of the 59 DSP slices to be multiplied by b_i which is also shifted one word to the right. This process accumulates all of the partial products. These partial products are then added together as previously described. After the final output is stored in memory; the multiplier is used again 158 more times to calculate all of the parts of x_i and the output is combined to achieve the final result.

Table 3 gives conservative estimates of timings for the multiplications required in the encryption step in all four versions of the FHE scheme in [10] without considering parallel implementation of multipliers, which would considerably speed up timings. We also assume a conservative estimate of a 500MHz clock frequency for the multiplier, as the critical path goes through the DSP block. The published software timings for the encryption step in [10] requires, for example, 1 second for the toy sized encryption step in the FHE scheme. The multiplication step is one of the two bottlenecks in the encryption scheme and this suggests that the use of hardware could greatly improve the practicality of such encryption steps or indeed any step in FHE schemes which requires large integer multiplication.

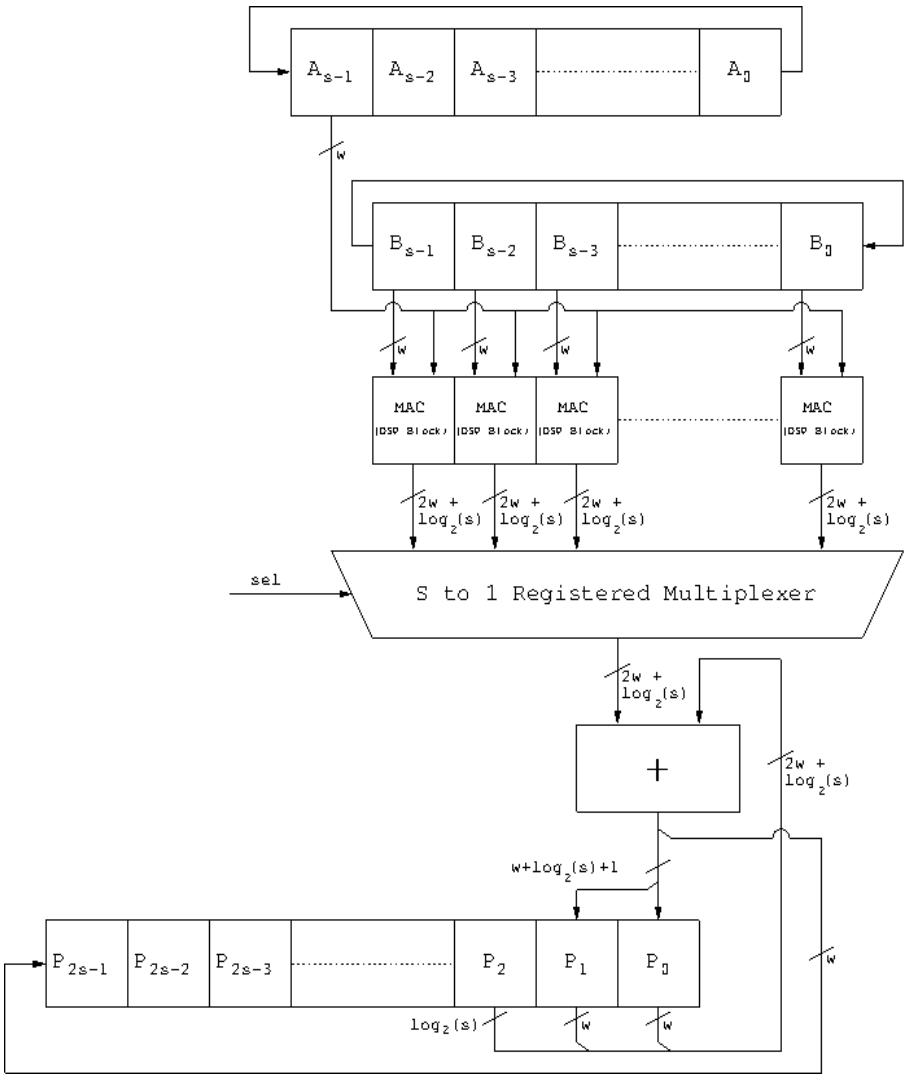

Fig. 1. Hardware Architecture of Comba Multiplier

We make the assumption that we can access the off-chip memory storage; storage of the products is an issue, especially with the larger versions of the FHE schemes but for the toy size this is not a major issue, as only 60 DSP slices are required per 944 bit multiplier. Moreover 158 of these 0.15Mbit-sized products require a total of 23.7 Mbits memory. This is manageable on the targeted Xilinx Virtex 7 XC7VX980T, as there is 68 Mbits block RAM (BRAM) available. For the large FHE scheme, each multiplication is around 19.35Mbits long and 7659 of these are required to be added,

which highlights the storage issues associated with the large scheme sizes. Additionally the transfer of data must also be considered. Not only do the parameter sizes increase with the larger versions of the FHE schemes but the number of required multiplications for encryption also increase; the issue of memory storage and access becomes a major issue and it is impossible to store the partial products or intermediate values within the memory storage on the FPGA. Obviously there is a need to make use of off-chip memory, which will require careful management so as not to become the architecture bottleneck.

Table 3. DSP Slices required and estimated timings for large integer multiplier in encryption step using Comba multiplication at 500 MHz

Size of Scheme:	**Toy**	**Small**	**Medium**	**Large**
No. of multiplications required in encryption τ	158	572	2110	7659
Size of required multiplier (bits)	936× 150000	1476× 830000	2016× 4200000	2556× 19350000
Target multiplier (bits)	944×944	1488×1488	2016×2016	2560× 2560
No. of DSP slices required for target multiplier	60	94	127	161
Estimation of Clock Cycles required (multiplications not run in parallel)	1507320	30002544	558449480	9320995341
Estimated timing of all multiplications required in encryption step (secs)	0.00301	0.06001	1.11690	18.64200
Published Timing (secs) of Encryption Step in [10]	0.05	1	21	435

We give an estimation of the timing for the multiplications required in each of these four versions using the number of multiplications required, an estimate of the number of required clock cycles to achieve the target multiplier size and the number of cycles required to achieve the full size multiplier. We do not consider parallelising the multiplications in this estimation, although this is possible, as the number of required DSP slices for the selected target multiplier for all four versions occupies less than 5% of the target FPGA DSP slices. Furthermore, we do not fully utilise the DSP slice multiplier of 18×25 bits; we could extend the 16×16 bit multiplier to a 17×24 bit unsigned multiplier for example, which would improve performance of the multiplier. Therefore Table 3 lists conservative estimates. From these results however, we can still see that the toy size version will fit on an FPGA and this could be parallelised to give an even better performance. Moreover the estimated timings for the small, medium and large schemes suggest that a hardware implementation of FHE could offer significant improvements to the practicality of FHE schemes. Pre-

liminary synthesis results* of a 944 bit multiplier show that it requires 59 DSP48E1s, and has a latency of 121 clocks: 1 for loading, 60 for multiply accumulate and 60 for partial product addition and shifting. The overall latency of the multiplier is $2s + 3$ clock cycles, where s is the number of words. To our knowledge, this is one of the first analyses into the practicality of an FPGA based implementation of crypto primitives for use in FHE schemes.

5 Conclusions

We have considered one of the most important building blocks involved in FHE schemes, large integer multiplication. We have looked at the Comba multiplication method and the possibility of targeting DSP48E1 slices on a Xilinx Virtex 7 FPGA to perform the large integer multiplication to ultimately improve the performance of FHE schemes. From the preliminary results we establish that the large integer multiplication in the encryption step for the toy scheme will fit comfortably on a single FPGA device. Furthermore the conservatively estimated timings suggest using a hardware implementation of this multiplication algorithm should improve performance of FHE schemes compared to the software implementations, especially for the larger versions of the FHE schemes [10]. This establishes the potential and justification for continuing research into hardware implementations of crypto-primitives, such as large integer multiplication, to improve the performance and hence the practicality of FHE schemes. There will however be issues with memory storage with these large versions of the FHE schemes. As this is a relatively recent area of research, there is a lot of future work still to be carried out and we are currently pursuing a hardware implementation of a complete encryption step of a FHE scheme.

References

1. Cloud Industry Forum: UK cloud adoption and trends for 2013, http://www.cloudindustryforum.org/white-papers/uk-cloud-adoption-and-trends-for-2013
2. El Gamal, T.: A Public Key Cryptosystem and a Signature Scheme based on Discrete Logarithms. IEEE Transactions on Information Theory. 31 (4), 473-481, (1985)
3. Paillier, P.: Public-Key Cryptosystems based on Composite Degree Residuosity Classes. In: EUROCRYPT. pp. 223-238 (1999)
4. Boneh, D., Goh, E., Nissim, K.: Evaluating 2-DNF Formulas on Ciphertexts. In: TCC. pp. 325-341 (2005)
5. Gentry, C.: A Fully Homomorphic Encryption Scheme. Stanford: PhD Dissertation (2009)
6. Brakerski, Z., Vaikuntanathan, V.: Efficient Fully Homomorphic Encryption from

* Post place and route results are not presented, due to over-mapping of the i/o pins

(standard) LWE. In: FOCS. pp. 97-106 (2011)

7. Gentry, C., Halevi, S., Smart, N. P.: Fully Homomorphic Encryption with Polylog Overhead. In: EUROCRYPT. pp. 465-482 (2012)
8. Gentry, C., Halevi, S., Smart, N. P.: Homomorphic Evaluation of the AES Circuit. In: CRYPTO. pp. 850-867 (2012)
9. Gentry, C., Halevi, S.: Implementing Gentry's Fully Homomorphic Encryption Scheme. In: EUROCRYPT. pp. 129-148 (2011)
10. Coron, J.S., Naccache, D., Tibouchi, M.: Public Key Compression and Modulus Switching for Fully Homomorphic Encryption over the Integers. In: EUROCRYPT. pp. 446-464 (2012)
11. van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully Homomorphic Encryption over the Integers. In: EUROCRYPT. pp. 24-43 (2010)
12. Coron, J.-S, Tibouchi, M., Naccache, D., Mandal, A.: Fully Homomorphic Encryption over the Integers with Shorter Public Keys. In: CRYPTO. pp. 487-504 (2011)
13. Cousins, D., Rohloff, K., Peikert, C., Schantz, R.: An update on SIPHER (Scalable Implementation of Primitives for Homomoprhic EncRyption) - FPGA implementation using Simulink. In: HPEC. pp.1-5 (2012)
14. Brenner, M., Perl, H. Smith, M.: Practical Applications of Homomorphic Encryption. In: SECRYPT. pp. 5-14 (2012)
15. Smart, N. P., Vercauteren, F.: Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes. In: PKC. pp. 420-443 (2010)
16. Göttert, N., Feller, T., Schneider, M., Buchmann, J., Huss, S. A.: On the Design of Hardware Building Blocks for Modern Lattice-Based Encryption Schemes. In: CHES. pp. 512-529 (2012)
17. Comba, P. G: Exponentiation Cryptosystems on the IBM PC. IBM Systems Journal. 29, 526-538 (1990)
18. Güneysu, T.: Utilizing Hard Cores of Modern FPGA Devices for High-Performance Cryptography. J. Cryptographic Engineering. 1, 37-55 (2011)
19. Bosselaers, A., Govaerts, R., Vandewalle, J.: Comparison of Three Modular Reduction Functions. In: CRYPTO. pp. 175-186 (1993)
20. Karatsuba, A., Ofman, Y.: Multiplication of Many-Digit Numbers by Automatic Computers. Doklady Akad. Nauk SSSR. 145, 293-294 (1962)
21. Rivest, R., Shamir, A., Adleman, L.: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, Communications of the ACM. 21, 120-126 (1978)
22. 7 Series FPGAs Overview, www.xilinx.com (Accessed 28 December 2012)
23. Brenner, M., Perl, H., Smith, M.: How Practical is Homomorphically Encrypted Program Execution? An Implementation and Performance Evaluation. In: TrustCom. pp. 375-382 (2012)